CompSci 372 S1 C 2006
Computer Graphics

Mid Term Test, 10th April 2006 – Sample Solution

Surname (Family Name):	
First Name(s):	
ID Number:	
Login Name (UPI):	

Instructions:

1. Attempt **ALL** questions.
2. The test is for one (1) hour.
3. This is a **closed book** test.
4. Calculators are **NOT** permitted.
5. Write your answers in the spaces provided. There is space at the back for answers that overflow the allotted space.
6. **Questions total 50 Marks**.
7. This test is worth 10% of your final marks for CompSci372 S1 C

Section	Marks	Maximum Marks
Q.1		**8**
Q.2		**10**
Q.3		**16**
Q.4		**10**
Q.5		**6**
Total		**50**

Question 1 – Short answer test [8 marks]

Complete each of the following statements by filling in the underlined blank spaces. Each blank space is worth 1 mark.

[8 marks]

(a) In order to render images of 3D scenes we can use polygon rendering, where polygons are projected onto the view plane, or **ray tracing** which requires us to trace for each pixel of the view plane a ray from the view point through the pixel into the scene.

(b) In this course we use three libraries for writing OpenGL programs: GL, GLU and **GLUT**.

(c) The technique for generating smooth surfaces from coarse control meshes by iteratively subdividing them is called **subdivision surface** .

(d) The technique for animating a model by recording movements of a human actor is called **motion capture**.

(e) Given are $\mathbf{u} = \begin{pmatrix} 3 \\ 1 \\ -2 \end{pmatrix}$ and $\mathbf{v} = \begin{pmatrix} -2 \\ 2 \\ 0 \end{pmatrix}$.

(i) The scalar product (dot product) of the two vectors is **u.v** = **3*(-2)+1*2+(-2)*0=-4**.

(ii) The vector product (cross product) of the two vectors is $\mathbf{u} \times \mathbf{v} = \begin{pmatrix} 1*0-(-2)*2 \\ (-2)*(-2)-3*0 \\ 3*2-1*(-2) \end{pmatrix} = \begin{pmatrix} 4 \\ 4 \\ 8 \end{pmatrix}$.

(f) Given are two 3D parallel vectors **a** and **b** which both have a length of 4 units.

Then **a•b** = **16** (your answer must be a number and not a formula).

(g) A ray with the direction **v** is reflected on a plane. The reflected vector is **w**. The normal of the plane is **n**= 0.5*(**w-v**) (or you can normalise your answer: $\frac{\mathbf{w}-\mathbf{v}}{|\mathbf{w}-\mathbf{v}|}$)

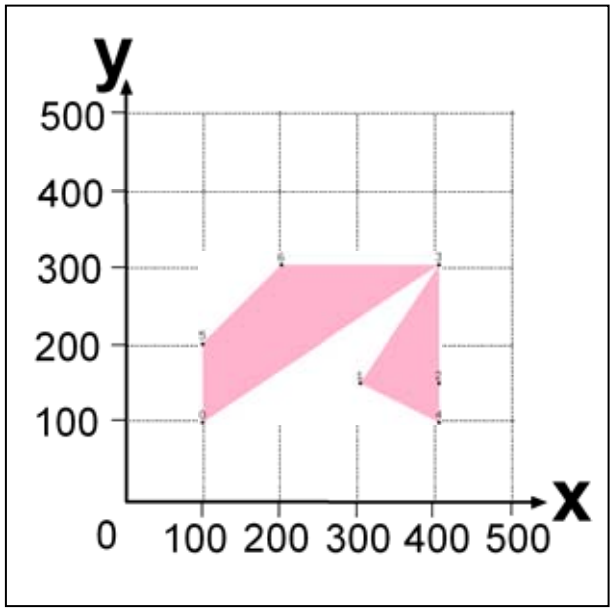
Question 2 – OpenGL [10 marks]

A. In this question you must draw the 2D objects which are generated for each sequence of OpenGL commands given below. For all parts of this question assume that the following data is defined:

```
const int numVertices=7;
const float vertices[numVertices][2] = {{100,100},{300,150},{400,150},
                                        {400,300},{400,100},{100,200},{200,300}};
```

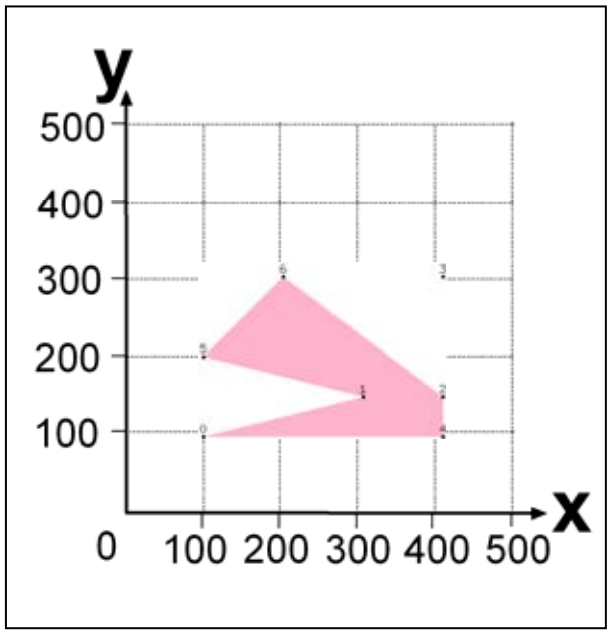
I. Sketch in the answer space on the right the objects drawn by the display method on the left [2 marks].

```
void display(void)
{
    // clear all pixels in frame buffer
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_QUADS);
    glVertex2fv(vertices[4]);
    glVertex2fv(vertices[2]);
    glVertex2fv(vertices[3]);
    glVertex2fv(vertices[1]);
    glVertex2fv(vertices[0]);
    glVertex2fv(vertices[3]);
    glVertex2fv(vertices[6]);
    glVertex2fv(vertices[5]);
    glEnd();
    glFlush();
}
```



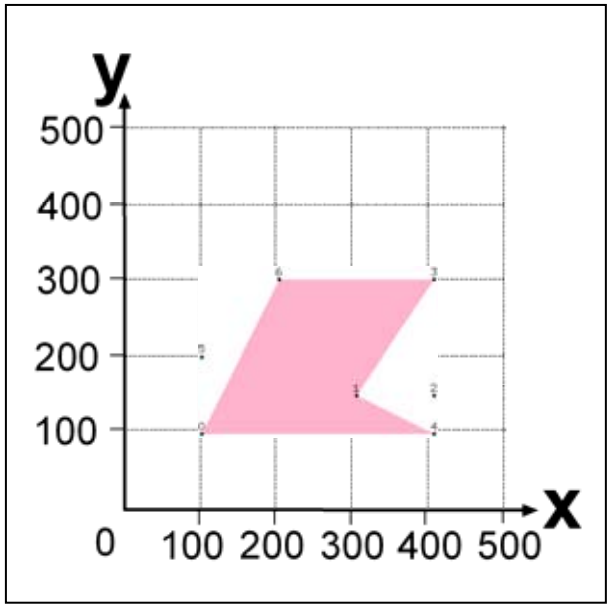
II. Sketch in the answer space on the right the objects drawn by the display method on the left [2 marks].

```
void display(void)
{
    // clear all pixels in frame buffer
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_QUAD_STRIP);
    glVertex2fv(vertices[0]);
    glVertex2fv(vertices[4]);
    glVertex2fv(vertices[1]);
    glVertex2fv(vertices[2]);
    glVertex2fv(vertices[5]);
    glVertex2fv(vertices[6]);
    glEnd();
    glFlush();
}
```



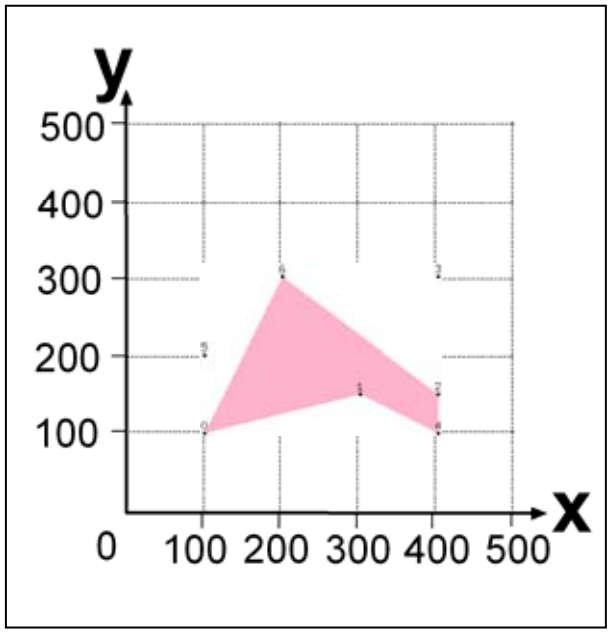
III. Sketch in the answer space on the right the objects drawn by the display method on the left [2 marks].

```
void display(void)
{
    // clear all pixels in frame buffer
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_TRIANGLE_STRIP);
    glVertex2fv(vertices[6]);
    glVertex2fv(vertices[3]);
    glVertex2fv(vertices[0]);
    glVertex2fv(vertices[1]);
    glVertex2fv(vertices[4]);
    glEnd();
    glFlush();
}
```



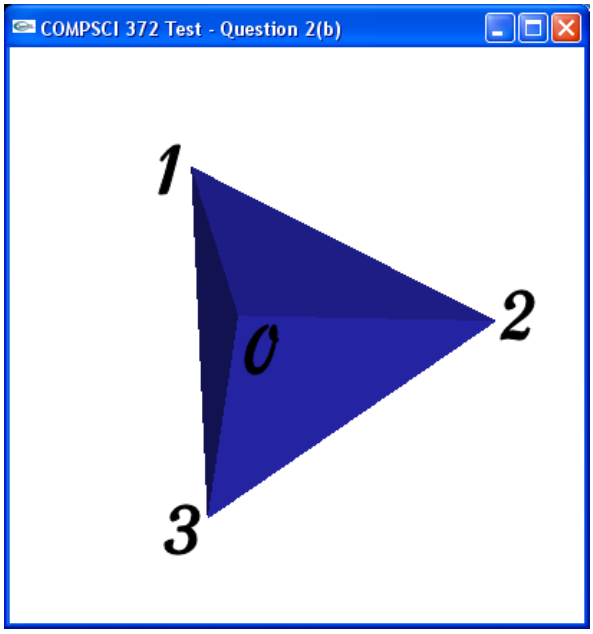
IV. Sketch in the answer space on the right the objects drawn by the display method on the left [2 marks].

```
void display(void)
{
    // clear all pixels in frame buffer
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_TRIANGLE_FAN);
    glVertex2fv(vertices[6]);
    glVertex2fv(vertices[0]);
    glVertex2fv(vertices[1]);
    glVertex2fv(vertices[4]);
    glVertex2fv(vertices[2]);
    glEnd();
    glFlush();
}
```



B. Given is an array of four 3D vertices "`const float v[4][3]`" defining the vertices of a tetrahedron as shown in the image below. Write the code to display this tetrahedron as efficient as possible using exactly **ONE** triangle strip. You do **not** have to define the normals of the tetrahedron's faces. [2 marks]

Note: You are allowed to define the triangle strip such that the vertices of some faces are in clockwise order and some in anti-clockwise order.




```
void display(void)
{
    glShadeModel(GL_FLAT);
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, blue_material);

    glBegin(GL_TRIANGLE_STRIP);
```

```
    glVertex3fv(v[1]);
    glVertex3fv(v[2]);
    glVertex3fv(v[0]);
    glVertex3fv(v[3]);
    glVertex3fv(v[1]);
    glVertex3fv(v[2]);
```

```
    glEnd();

    glFlush ();
    glutSwapBuffers();
}
```

Question 3 – 2D and 3D Geometry [16 marks]

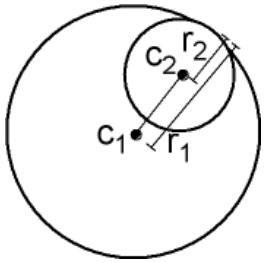
A. Given are two spheres with the centres $\mathbf{c}_1$ and $\mathbf{c}_2$ and the radii r_1 and r_2, respectively. Assume the centre of one sphere lies **inside** the other sphere. Write in the answer space below a formula for testing whether the two spheres intersect.

[2 mark]

By drawing a little picture we can see easily that the two spheres just touch when the distance between the centres is equal to the difference in radii.

Hence our test is: $|\mathbf{c}_1-\mathbf{c}_2| \geq |r_1-r_2|$

The two spheres intersect if the above expression is true.



B. Write a C/C++ function which takes as input the three 3D vertices $\mathbf{p}_0$, $\mathbf{p}_1$, and $\mathbf{p}_2$ of a triangle and the direction $\mathbf{v}$ of a ball and which returns the direction of the ball after it is reflected on the triangle. You may assume that the ball hits the face of the triangle (i.e. you can ignore reflections on the edges and corners of the triangle). [4 mark]

You are allowed to use the functions and classes defined in Appendix A.

```
CVec3df reflectionOnTriangle(CVec3df& v, CVec3df& p0, CVec3df& p1, CVec3df& p2)
{
  CVec3df e0=p1-p0;          // compute edges of triangle
  CVec3df e1=p2-p0;
  CVec3df n=cross(e0,e1);    // compute normal of triangle
  n.normaliseDestructive(); // normalise it
  return v-2*dot(v,n)*n;     // use reflection formula from
                             // the lecture notes
}
```

C. Given is a plane $\begin{pmatrix}1\\-2\\2\end{pmatrix}\bullet\mathbf{p}=3$ and a triangle with the vertices $\mathbf{p}_0=\begin{pmatrix}2\\0\\1\end{pmatrix}$, $\mathbf{p}_1=\begin{pmatrix}0\\-2\\0\end{pmatrix}$ and $\mathbf{p}_2=\begin{pmatrix}3\\1\\-1\end{pmatrix}$. Does the triangle intersect the plane? Proof your answer!

[5 marks]

<u>Solution 1:</u> Find a point on the plane and use the Inside-Outside Half-Space test from handout 6, slide 26:

The point S=(3,0,0) fulfills the plane equation and is hence on the plane. Now we test for each point P_i whether the angle between (P_i-S) and n is greater or smaller than 90 degree:

$$(P_0-S)\bullet\mathbf{n}=\begin{pmatrix}-1\\0\\1\end{pmatrix}\bullet\begin{pmatrix}1\\-2\\2\end{pmatrix}=1>0\quad\Rightarrow\text{Point outside the half - space}$$

$$(P_1-S)\bullet\mathbf{n}=\begin{pmatrix}-3\\-2\\0\end{pmatrix}\bullet\begin{pmatrix}1\\-2\\2\end{pmatrix}=1>0\quad\Rightarrow\text{Point outside the half - space}$$

$$(P_2-S)\bullet\mathbf{n}=\begin{pmatrix}0\\1\\-1\end{pmatrix}\bullet\begin{pmatrix}1\\-2\\2\end{pmatrix}=-4<0\quad\Rightarrow\text{Point inside the half - space}$$

Hence at least one vertex of the triangle lies on either side of the plane and hence the triangle must intersect the plane.

<u>Solution 2:</u> An even easier solution is to compute for each vertex of the triangle its distance to the plane (handout 6, slide 25) – if it is negative the point lies inside, otherwise outside the plane. Note that in this case we are only interested in the sign of the distance and hence can omit normalizing the plane equation.

$$\mathbf{p}_0\bullet\mathbf{n}-d=\begin{pmatrix}2\\0\\1\end{pmatrix}\bullet\begin{pmatrix}1\\-2\\2\end{pmatrix}-3=1>0\quad\Rightarrow\text{Point outside}$$

$$\mathbf{p}_1\bullet\mathbf{n}-d=\begin{pmatrix}0\\-2\\0\end{pmatrix}\bullet\begin{pmatrix}1\\-2\\2\end{pmatrix}-3=1>0\quad\Rightarrow\text{Point outside}$$

$$\mathbf{p}_2\bullet\mathbf{n}-d=\begin{pmatrix}3\\1\\-1\end{pmatrix}\bullet\begin{pmatrix}1\\-2\\2\end{pmatrix}-3=-4<0\quad\Rightarrow\text{Point inside}$$

<u>Solution 3:</u> Test whether any of the three edges of the triangle intersects the plane.

Test edge P_0P_1: The line equation is

$$\mathbf{p}(t) = \mathbf{p}_0 + t(\mathbf{p}_1 - \mathbf{p}_0) = \mathbf{p}_0 + \mathrm{t}\mathbf{v} = \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix} + \mathrm{t}\begin{pmatrix} -2 \\ -2 \\ -1 \end{pmatrix}, t \in [0,1]$$

However, since $\mathbf{n} \bullet \mathbf{v} = \begin{pmatrix} 1 \\ -2 \\ 2 \end{pmatrix} \bullet \begin{pmatrix} -2 \\ -2 \\ -1 \end{pmatrix} = 0$ **the edge is parallel to the plane and no intersection point exists.**

Test edge P_0P_2: The line equation is

$$\mathbf{p}(t) = \mathbf{p}_0 + t(\mathbf{p}_2 - \mathbf{p}_0) = \mathbf{p}_0 + \mathrm{t}\mathbf{v} = \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix} + \mathrm{t}\begin{pmatrix} 1 \\ 1 \\ -2 \end{pmatrix}, t \in [0,1]$$

Inserting this into the plane equation gives:

$$\mathbf{n} \bullet \mathbf{p}(t) = d$$
$$\Leftrightarrow \mathbf{n} \bullet (\mathbf{p}_0 + t\mathbf{v}) = d$$
$$\Leftrightarrow \mathbf{n} \bullet \mathbf{p}_0 + t\mathbf{n} \bullet \mathbf{v} = d$$
$$\Leftrightarrow t = \frac{d - \mathbf{n} \bullet \mathbf{p}_0}{\mathbf{n} \bullet \mathbf{v}} = \frac{3-4}{-5} = 0.2$$

Hence the edge intersects the plane and therefore the triangle intersects the plane.

D. Given are two lines in 2D with the parametric equations $\mathbf{p}(t) = \begin{pmatrix} 2 \\ 3 \end{pmatrix} + t \begin{pmatrix} 2 \\ -1 \end{pmatrix}$ and $\mathbf{q}(s) = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + s \begin{pmatrix} -1 \\ 1 \end{pmatrix}$, respectively. Compute the intersection point of these two lines.

[5 marks]

We have to find s,t such that $\mathbf{p}(t)=\mathbf{q}(s)$:

$$\mathbf{p}(t) = \mathbf{q}(s) \Leftrightarrow \begin{pmatrix} 2 \\ 3 \end{pmatrix} + t \begin{pmatrix} 2 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + s \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

$$\Leftrightarrow t \begin{pmatrix} 2 \\ -1 \end{pmatrix} - s \begin{pmatrix} -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} - \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

$$\Leftrightarrow \begin{pmatrix} 2 & 1 \\ -1 & -1 \end{pmatrix} \begin{pmatrix} t \\ s \end{pmatrix} = \begin{pmatrix} -2 \\ -2 \end{pmatrix}$$

$$\Leftrightarrow \begin{pmatrix} t \\ s \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ -1 & -1 \end{pmatrix}^{-1} \begin{pmatrix} -2 \\ -2 \end{pmatrix} = \frac{1}{-1} \begin{pmatrix} -1 & -1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} -2 \\ -2 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ -1 & -2 \end{pmatrix} \begin{pmatrix} -2 \\ -2 \end{pmatrix} = \begin{pmatrix} -4 \\ 6 \end{pmatrix}$$

Hence the intersection point is $\mathbf{p}(-4) = \mathbf{q}(6) = \begin{pmatrix} -6 \\ 7 \end{pmatrix}$

NOTE: When dealing with a simple system of equations it is much easier to solve it directly without using the inverse matrix, e.g.

$$\mathbf{p}(t) = \mathbf{q}(t) \begin{cases} (1)\ 2 + 2t = -s \\ (2)\ 3 - t = 1 + s \end{cases}$$

Adding equations $(1) + (2)$ gives :

$5 + t = 1 \Rightarrow t = -4$

Inserting this into equation (2) gives : $s = 3 - t - 1 = 6$

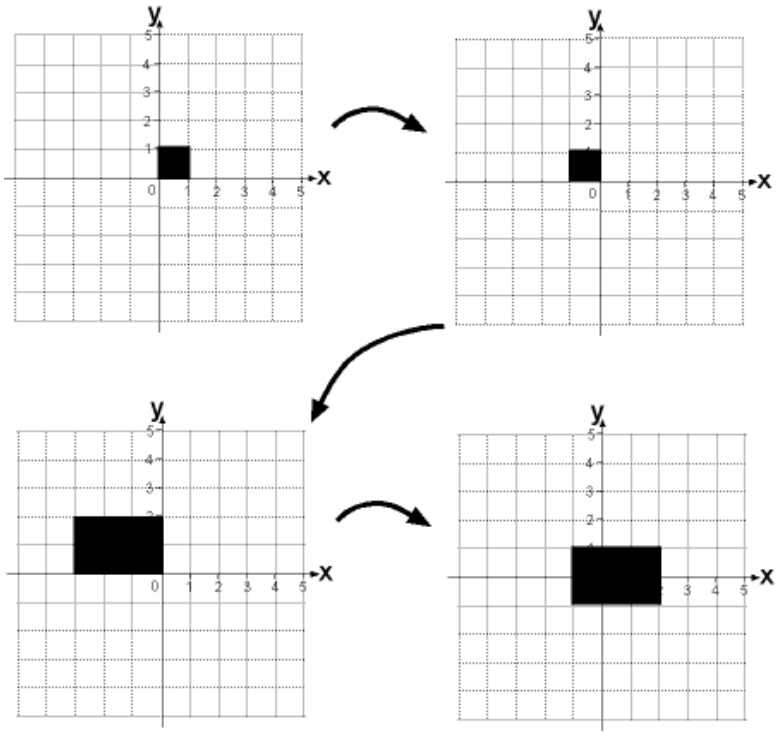
Question 4 – Transformations [10 marks]

A. Given is a polygon with the vertices (0,0), (0,1), (1,0) and (1,1). Draw in the answer space below the shape obtained after transforming this unit square with the homogeneous matrix

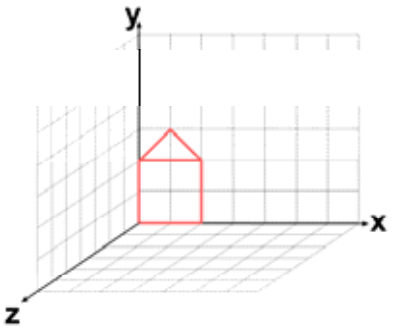
$$\mathbf{M} = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(90) & -\sin(90) & 0 \\ \sin(90) & \cos(90) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

[3 marks]

The matrix **M** first rotates the object by 90 degree, then scales it by a factor of 3 in x-direction and 2 in y-direction and then translates it by 2 in x-direction and -1 in y-direction. Hence the resulting image is:

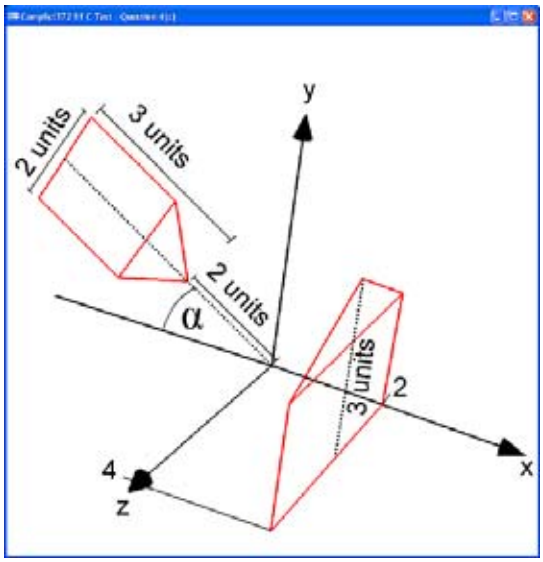


B. Given is a function `drawHouse()` which draws a wire frame house in the xy-plane with width 2 and height 3 as shown in the image on the right.



Use the function `drawHouse()` and OpenGL transformations in order to draw the scene show in the image on the right. Assume you have a variable `alpha` which defines the angle α.

[7 marks]




```
void display(void)
{
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();       // set view
    gluLookAt(0,0,20, 0,1.25,0, 0,1,0);
    trackball.tbMatrix();   // rotate the scene using the trackball ...

    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    float alpha=someValue;    // assume this value is defined elsewhere in the code
```

```
// draw the house in the xz-plane
glPushMatrix();
glRotatef(-(90+alpha),0,0,1);  // 2.Rotate by (90+alpha)
                               // in clockwise direction
glTranslatef(-1,-5,0);// 1.Translate house such that the
                      // tip of the roof is at (0,-2,0)
drawHouse();
glPopMatrix();

// draw the house parallel to the yz-plane
glTranslatef(2,0,0);     // 3.Translate the house by
                         // two units in x-direction
glRotatef(-90,0,1,0);    // 2.Rotate such that the house
                         // lies in the yz-plane
glScalef(2,1,1);         // 1.Scale width of house by 2
drawHouse();
```

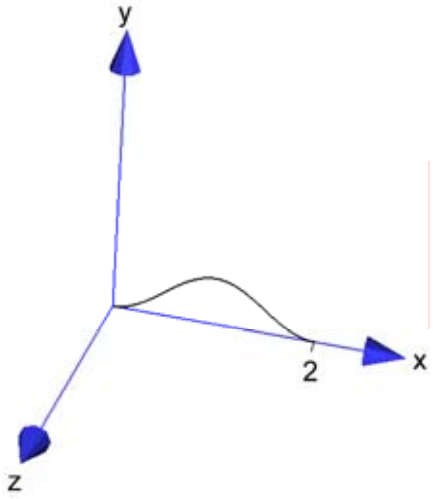
```
    glFlush ();
    glutSwapBuffers();
}
```

Question 5 – Modelling (Curves and Surfaces) [6 marks]

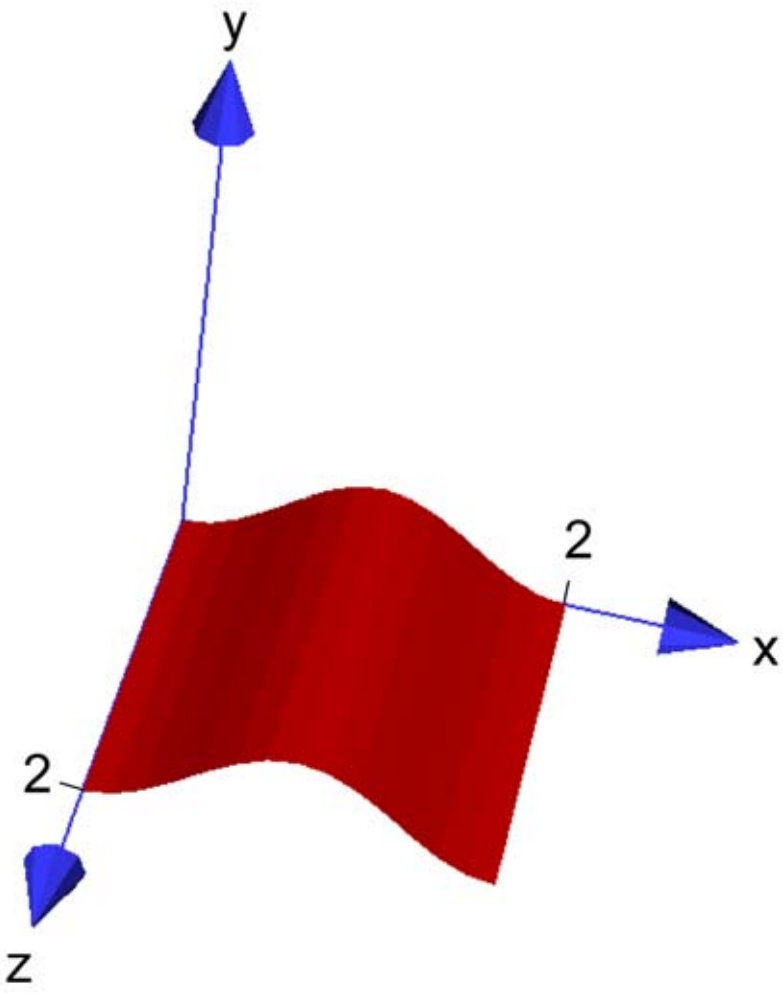
A. Given is a the function

```
float* curve(float t);
```

which returns an array of three floats representing the points of the parametric curve shown in the image on the right. The parameter *t* of the function is $0 \le t \le 1$ and $t=0$ gives the curve point at the origin of the coordinate system.



Write a function for displaying the surface shown in the image below. The profile of the surface is the above curve. **Note that the surface is flat shaded and you have to compute for every polygon of the surface one surface normal**.
You are allowed to use the functions and classes in appendix A.



```
void display(void)
{
   glMatrixMode( GL_MODELVIEW );  // Set the view matrix ...
   glLoadIdentity();              // ... to identity.
   gluLookAt(0,5,20, 1,0,0, 0,1,0); // camera is on the z-axis
   trackball.tbMatrix();          // rotate the cylinder using the trackball ...

   glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mat_ambient_and_diffuse);
   glShadeModel(GL_FLAT);

   // Draw the surface
   int nSteps=15;   // number of steps for subdividing the curve
```

```
float *c,*c1,t,t1;
glBegin(GL_QUAD_STRIP);   // Draw a quadstrip along the curve
for(int i=0;i<nSteps;i++)
{
   t=(float) i/(nSteps-1);
   c=curve(t);
   glVertex3f(c[0],c[1],0);  // Each segment of the quadstrip
   glVertex3f(c[0],c[1],2);  // forms edge parallel to z-axis
   t1=(float) (i+1)/(nSteps-1);
   c1=curve(t1);
   CVec3df v1(c1[0]-c[0],c1[1]-c[1],0);
   CVec3df z(0,0,1);          // Compute normal by taking the
   CVec3df n=cross(z,v1);     // cross product of the edges
                              // of each polygon of quadstrip
   n.normaliseDestructive();
   glNormal3fv(n.getArray());
}
glEnd();
```

```
   glFlush ();
   glutSwapBuffers();
}
```

Appendix A

```
class CVec3df {
   public:
       // Constructors/ Destructor
       CVec3df();
       CVec3df(float x, float y, float z);
       CVec3df(const CVec3df& v); // Copy constructor
       virtual ~CVec3df();

       // Assignment operator
       CVec3df& operator=(const CVec3df& v1);

       // Vector in array form
       float* getArray() { return v;}

       // some other operators
       CVec3df& operator+=(const CVec3df& v1);
       CVec3df& operator-=(const CVec3df& v1);
       CVec3df& operator*=(float scalar);
       CVec3df& operator/=(float scalar);

       friend CVec3df operator+(const CVec3df& v1, const CVec3df& v2);
       friend CVec3df operator-(const CVec3df& v1, const CVec3df& v2);
       friend CVec3df operator*(float scalar, const CVec3df& v1);
       friend CVec3df operator*(const CVec3df& v1, float scalar);
       friend CVec3df operator/(const CVec3df& v1, float scalar);
       friend CVec3df operator*(const CVec3df& v1, const CVec3df& v2);
       friend CVec3df operator-(const CVec3df& v1);
       friend bool operator==(const CVec3df& v1, const CVec3df& v2);
       friend bool operator!=(const CVec3df& v1, const CVec3df& v2);

       // normalize
       CVec3df normalise(void) const;    // returns a normalised vector
       void normaliseDestructive(void);  // normalises the calling vector object

       float dot(const CVec3df& v1) const;     // dot product
       CVec3df cross(const CVec3df& v1) const; // cross product
   private:
       float* v;
};

// more convenient way to use the dot and cross products
inline float dot(const CVec3df& v1, const CVec3df& v2) { return v1.dot(v2); }
inline CVec3df cross(const CVec3df& v1, const CVec3df& v2) { return v1.cross(v2); }
```